OIPE JC68 SEP 27 2004 PATENT & TRADEMARK OFFICE

FIG. 1

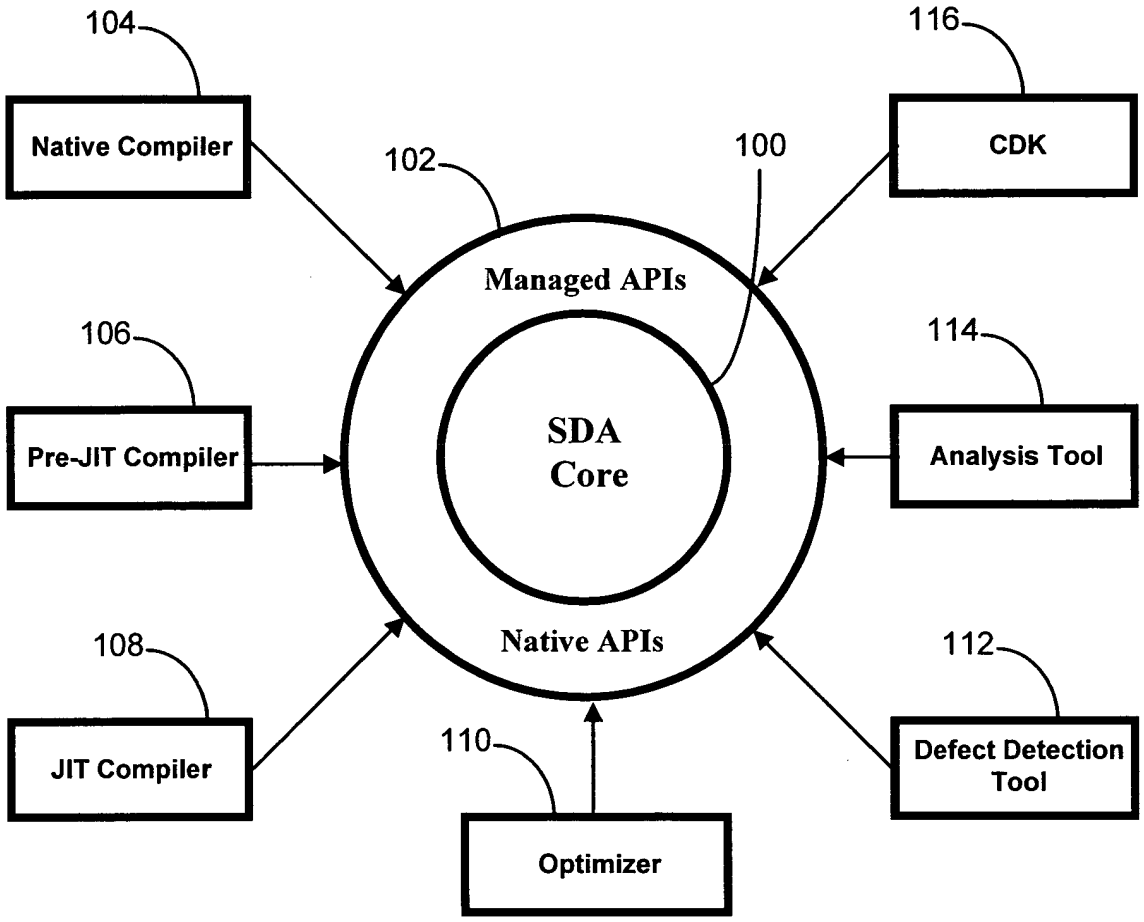


FIG. 2(a)

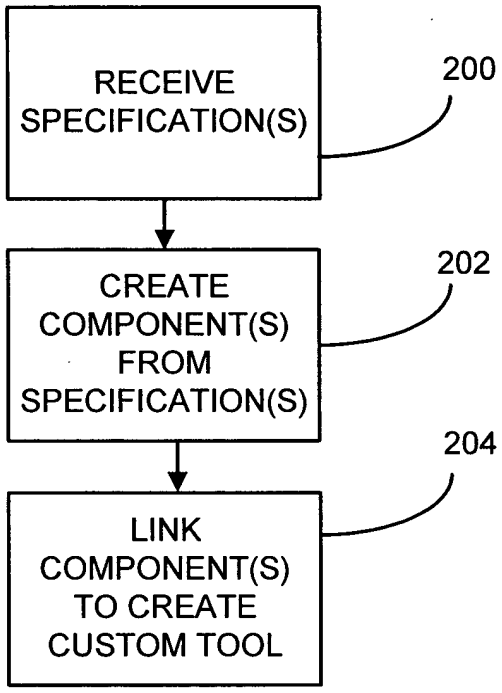
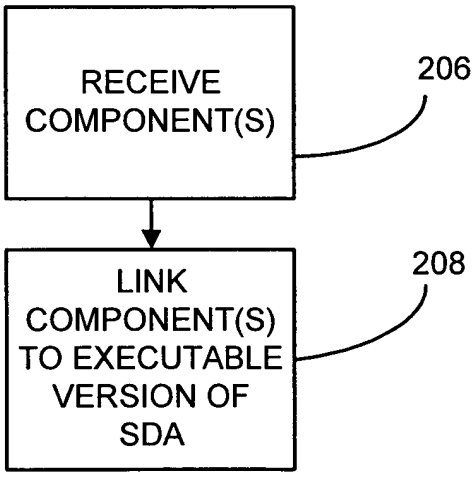

RECEIVE SPECIFICATION(S)
200
CREATE COMPONENT(S) FROM SPECIFICATION(S)
202
LINK COMPONENT(S) TO CREATE CUSTOM TOOL
204

FIG. 2(b)


RECEIVE COMPONENT(S)
206
LINK COMPONENT(S) TO EXECUTABLE VERSION OF SDA
208

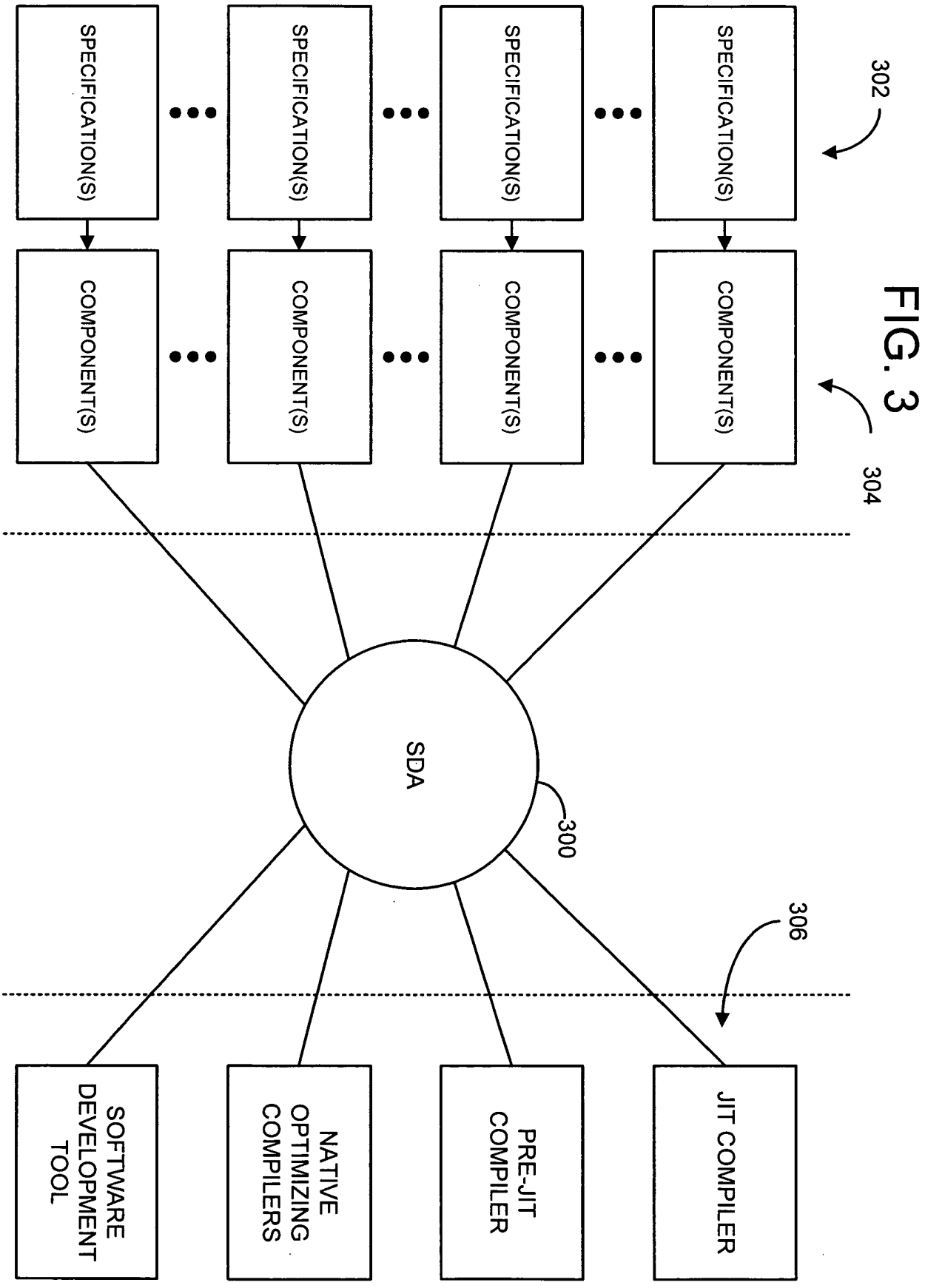

SPECIFICATION(S)
SPECIFICATION(S)
SPECIFICATION(S)
SPECIFICATION(S)
302
COMPONENT(S)
COMPONENT(S)
COMPONENT(S)
COMPONENT(S)
304
SDA
300
306
SOFTWARE DEVELOPMENT TOOL
NATIVE OPTIMIZING COMPILERS
PRE-JIT COMPILER
JIT COMPILER

FIG. 3

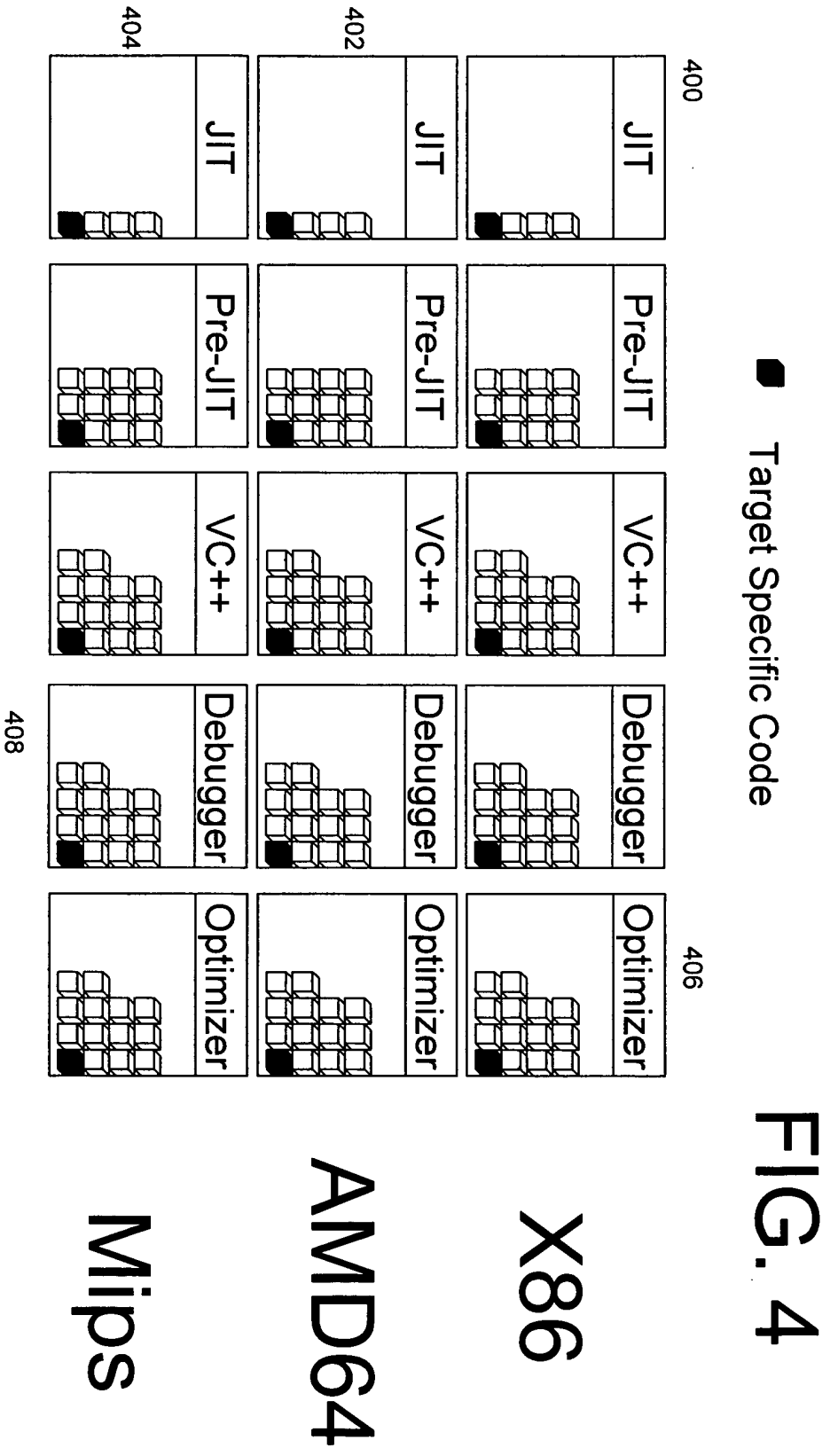

Target Specific Code
400
402
404
406
408
JIT
Pre-JIT
VC++
Debugger
Optimizer
X86
AMD64
Mips

FIG. 4

FIG. 5

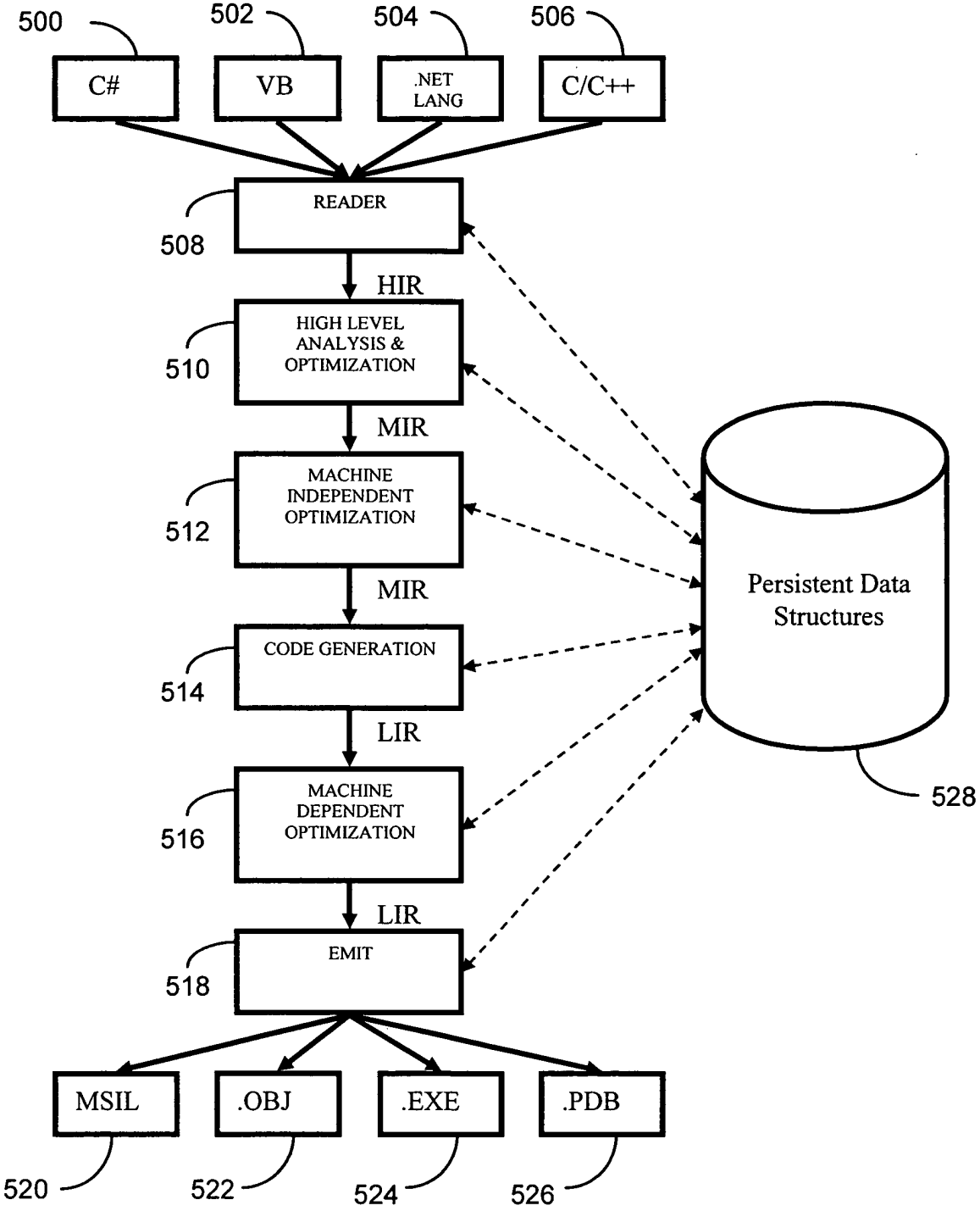

500
502
504
506
C#
VB
.NET LANG
C/C++
READER
508
HIR
HIGH LEVEL ANALYSIS & OPTIMIZATION
510
MIR
MACHINE INDEPENDENT OPTIMIZATION
512
MIR
CODE GENERATION
514
LIR
MACHINE DEPENDENT OPTIMIZATION
516
LIR
EMIT
518
Persistent Data Structures
528
MSIL
.OBJ
.EXE
.PDB
520
522
524
526

FIG. 6(a)

Source

```
int foo(int a, int b)
{
   int r;

   if (a < b)
   {
      r = a + 1;
   }
   else
   {
      r = b + 1;
   }

   return r;
```

FIG. 6(b)

Dump of high-level, machine independent IR

```
   _a.i32, _b.i32     = ENTER _foo                          #4
   t107.cond          = CMP(LT) _a.i32, _b.i32              #7
                        CBRANCH(LT) t107.cond, L2, L1       #7
L2:                                                         #7
   t108.i32           = ADD _a.i32, 1.i32                   #9
   _r.i32             = ASSIGN t108.i32                     #9
                        GOTO L3                             #11
L1:                                                         #7
   t109.i32           = ADD _b.i32, 1.i32                   #13
   _r.i32             = ASSIGN t109.i32                     #13
                        GOTO L3                             #11
L3:                                                         #11
                        RETURN _r.i32                       #16
                        GOTO L4                             #16
L4:                                                         #16
                        EXIT _foo                           #17
```

FIG. 6(c)

Dump of high-level with SSA, machine independent IR

Explicit wiring of SSA graph using definition numbers shown in <#> blue.

```
<1>_a.i32, <2>_b.i32     = ENTER _foo                                    #4
   <3>t107.cond          = CMP(LT) <1>_a.i32, <2>_b.i32                  #7
                           CBRANCH(LT) <3>t107.cond, L2, L1              #7
L2:                                                                      #7
   <4>t108.i32           = ADD <1>_a.i32, 1.i32                          #9
   <5>_r.i32             = ASSIGN <4>t108.i32                            #9
                           GOTO L3                                       #11
L1:                                                                      #7
   <6>t109.i32           = ADD <2>_b.i32, 1.i32                          #13
   <7>_r.i32             = ASSIGN <6>t109.i32                            #13
                           GOTO L3                                       #11
L3:                                                                      #11
   <8>_r.i32             = PHI <5>_r.i32, <7>_r.i32                      #16
                           RETURN <8>_r.i32                              #16
                           GOTO L4                                       #16
L4:                                                                      #16
                           EXIT _foo                                     #17
```

FIG. 6(d)

Dump of low-level, machine dependent IR (target X86)

```
    _a.i32, _b.i32     = ENTER _foo                                      #4
    {ESP}              = push EBP.i32, {ESP}                             #4
    EBP.i32            = mov ESP.i32                                     #4
    ESP.up32->unk, EFLAGS.cc32 = sub ESP.up32->unk, 4.i32                #4
                         PROLOGEND                                       #4
    t110(EAX).i32      = mov _b[EBP.up32->unk].i32.a32                   #7
    t107(EFLAGS).cond = cmp(LT) _a[EBP.up32->unk].i32.a32, t110(EAX).i32 #7
                         jge(GE) t107(EFLAGS).cond, L1                   #7
L2:                                                                      #7
    tv108-(EAX).i32    = mov 1.i32                                       #9
    tv108-(EAX).i32, EFLAGS.cc32 = add tv108-(EAX).i32, _a[EBP.up32-
>unk].i32.a32
#9
    _r[EBP.up32->unk].i32.a32 = mov tv108-(EAX).i32                      #9
                         jmp L3                                          #11
L1:                                                                      #7
    tv109-(EAX).i32    = mov 1.i32                                       #13
    tv109-(EAX).i32, EFLAGS.cc32 = add tv109-(EAX).i32, _b[EBP.up32-
>unk].i32.a32
#13
    _r[EBP.up32->unk].i32.a32 = mov tv109-(EAX).i32                      #13
L3:                                                                      #11
    t113(EAX).i32      = mov _r[EBP.up32->unk].i32.a32                   #16
L4:                                                                      #16
                         EPILOGSTART                                     #17
    ESP.i32            = mov EBP.i32                                     #17
    EBP.i32, {ESP}     = pop {ESP}                                       #17
    {ESP}              = ret {ESP}                                       #17
                         EXIT _foo, t113(EAX).i32                        #17
```

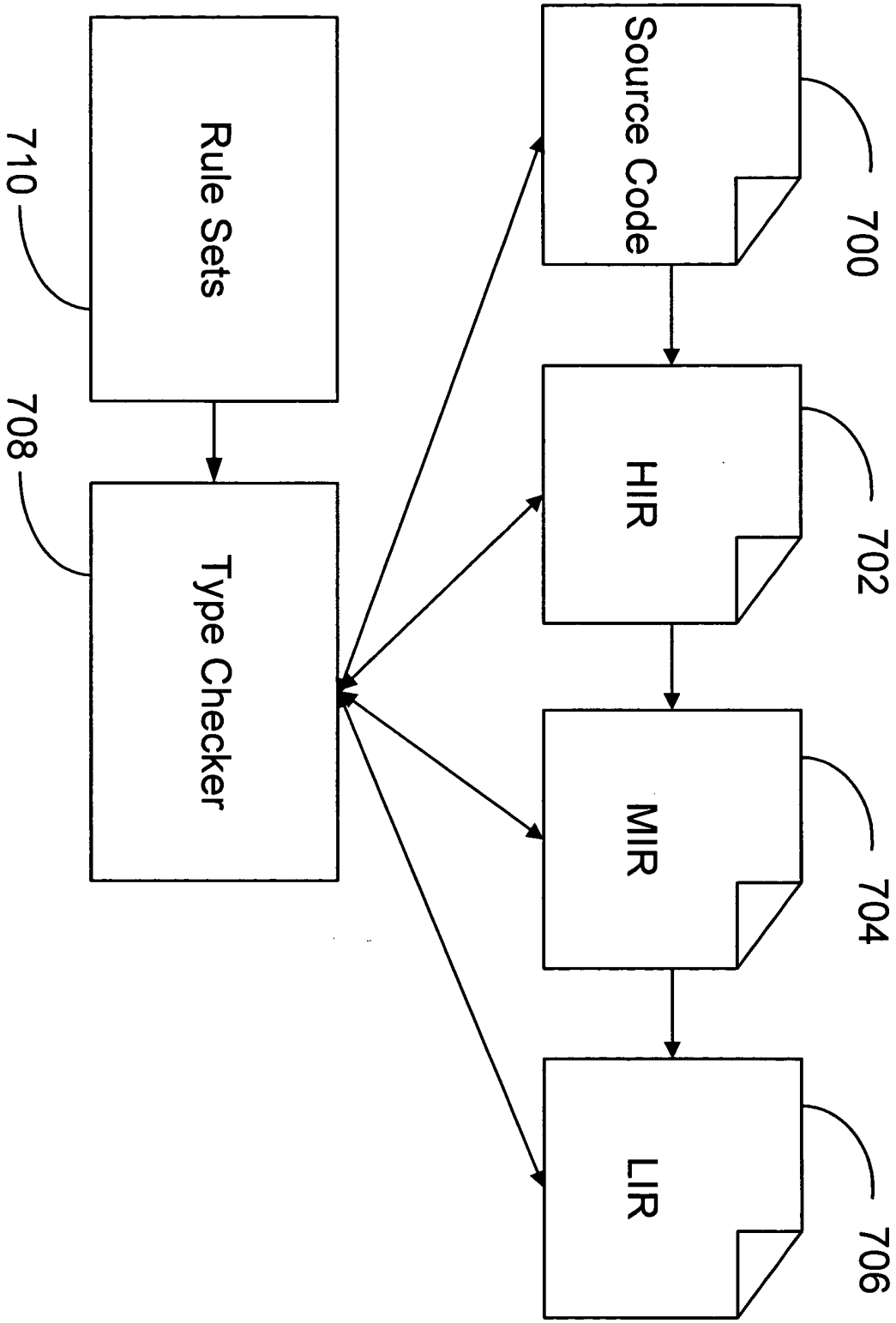

Source Code
700
Rule Sets
710
HIR
702
Type Checker
708
MIR
704
LIR
706

FIG. 7

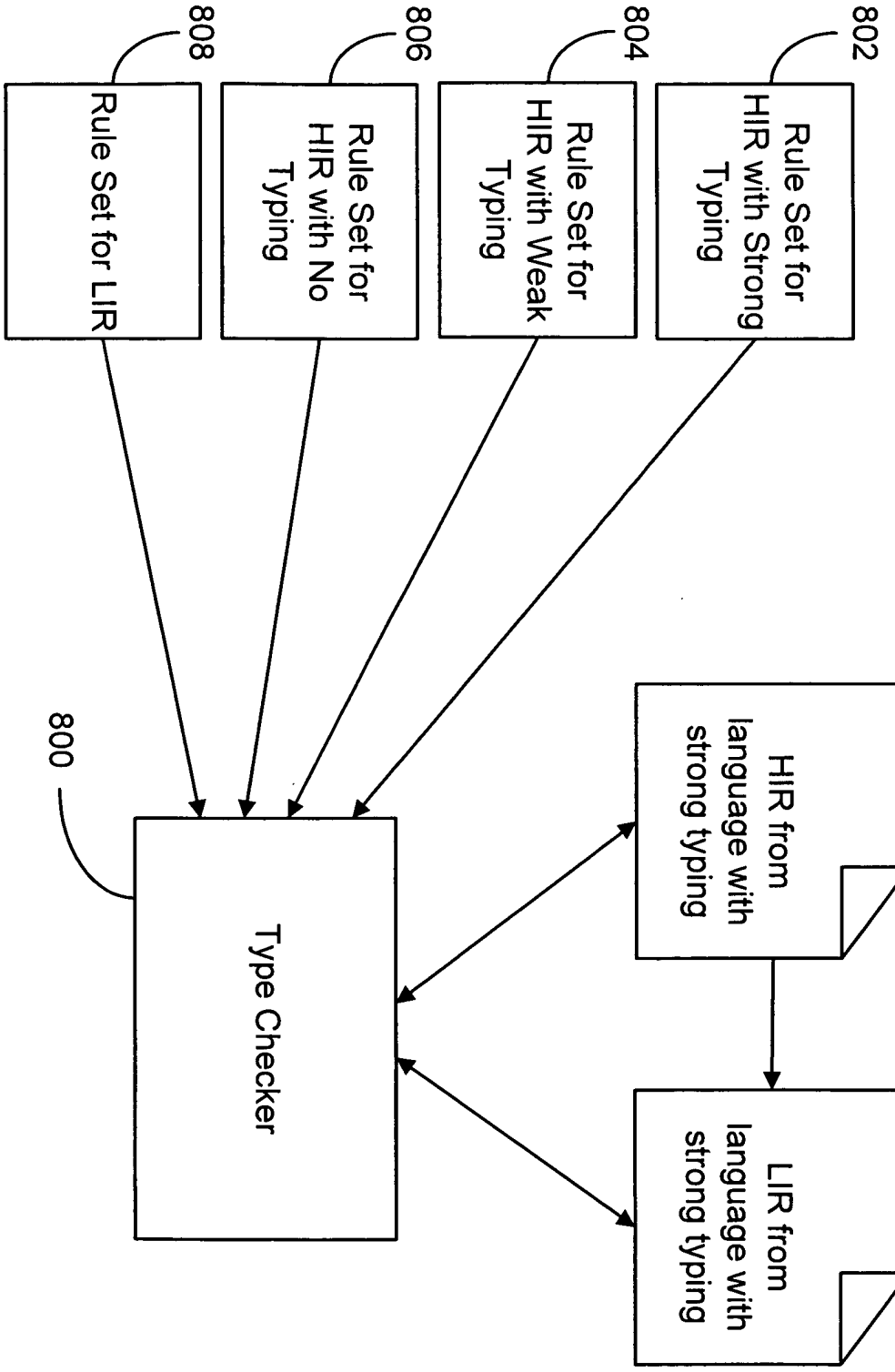

802
Rule Set for HIR with Strong Typing
804
Rule Set for HIR with Weak Typing
806
Rule Set for HIR with No Typing
808
Rule Set for LIR
800
Type Checker
810
HIR from language with strong typing
812
LIR from language with strong typing

FIG. 8

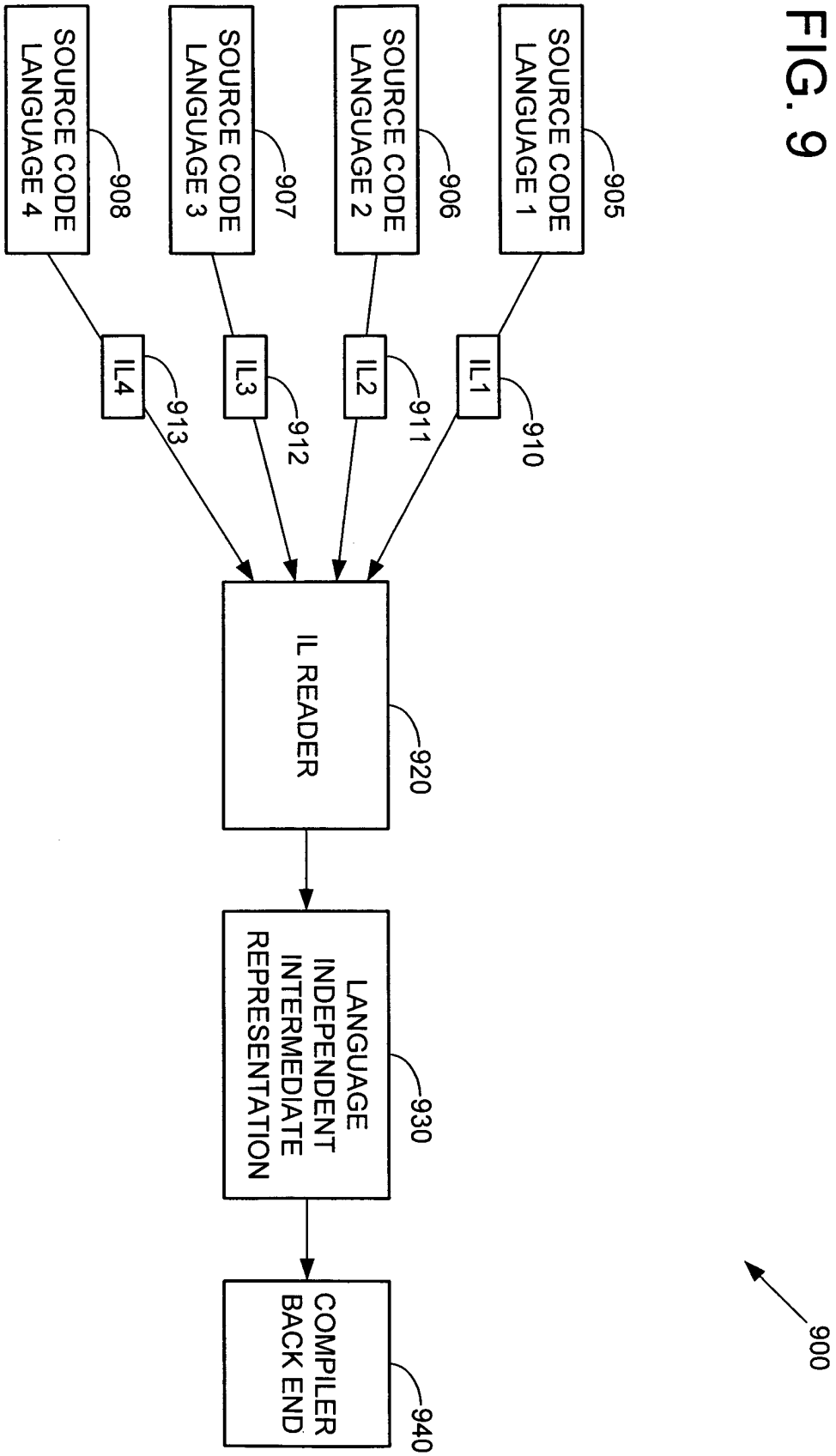

SOURCE CODE LANGUAGE 4
908
SOURCE CODE LANGUAGE 3
907
SOURCE CODE LANGUAGE 2
906
SOURCE CODE LANGUAGE 1
905
IL4
913
IL3
912
IL2
911
IL1
910
IL READER
920
LANGUAGE INDEPENDENT INTERMEDIATE REPRESENTATION
930
COMPILER BACK END
940
900

FIG. 9

FIG. 10A

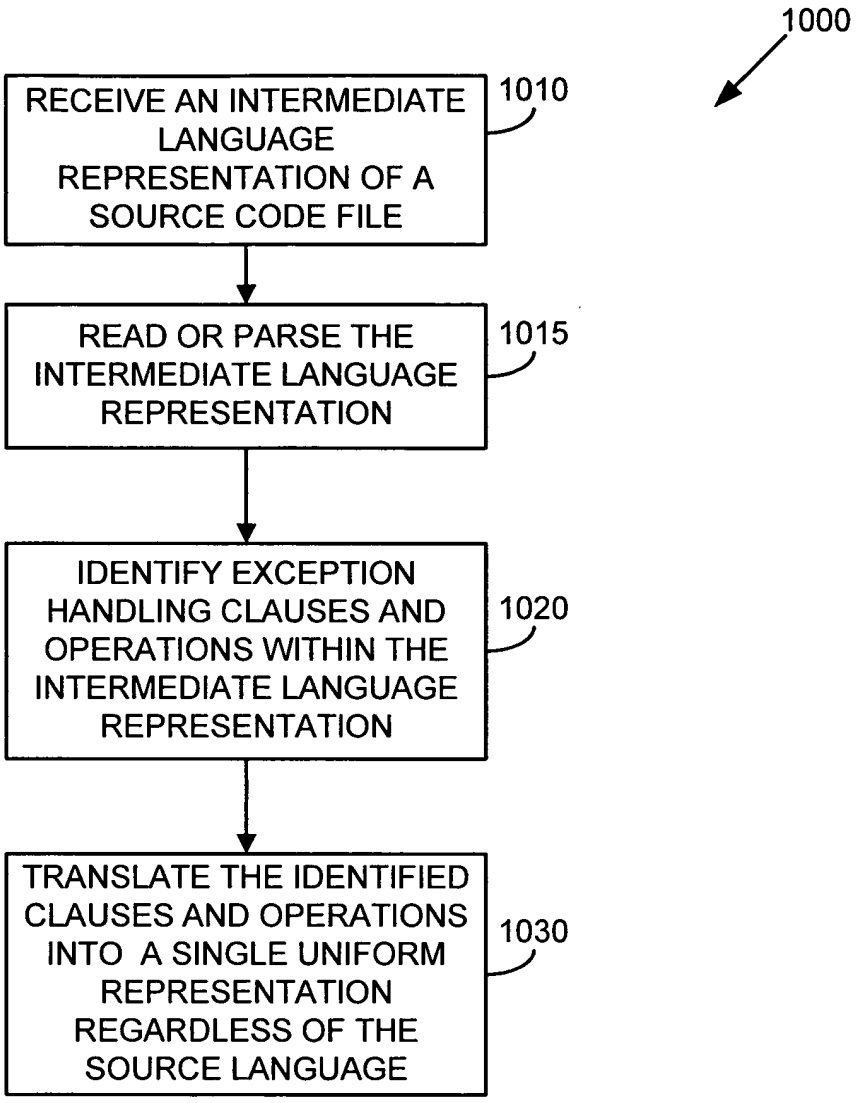
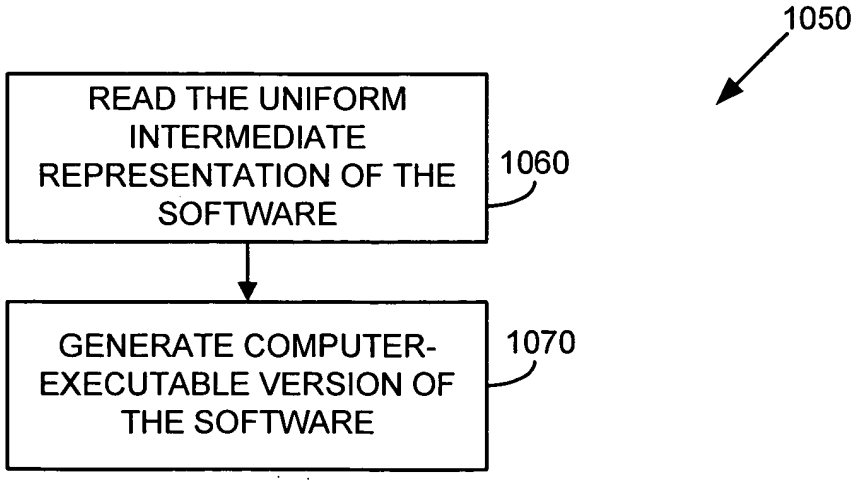


FIG. 10B



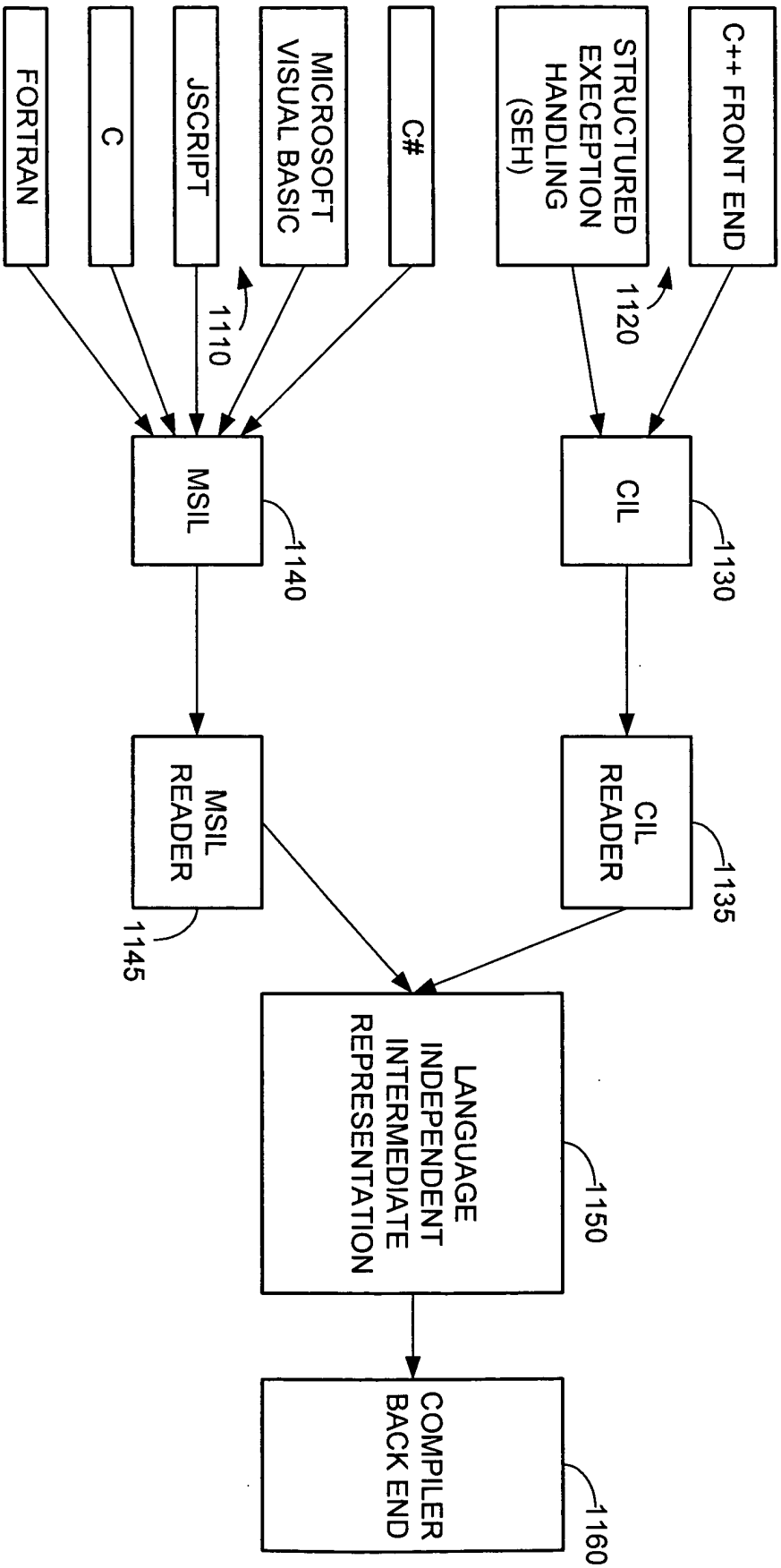

FORTRAN
C
JSCRIPT
MICROSOFT VISUAL BASIC
C#
1110
MSIL
1140
MSIL READER
1145
STRUCTURED EXECEPTION HANDLING (SEH)
C++ FRONT END
1120
CIL
1130
CIL READER
1135
LANGUAGE INDEPENDENT INTERMEDIATE REPRESENTATION
1150
COMPILER BACK END
1160

FIG. 11

FIG. 12

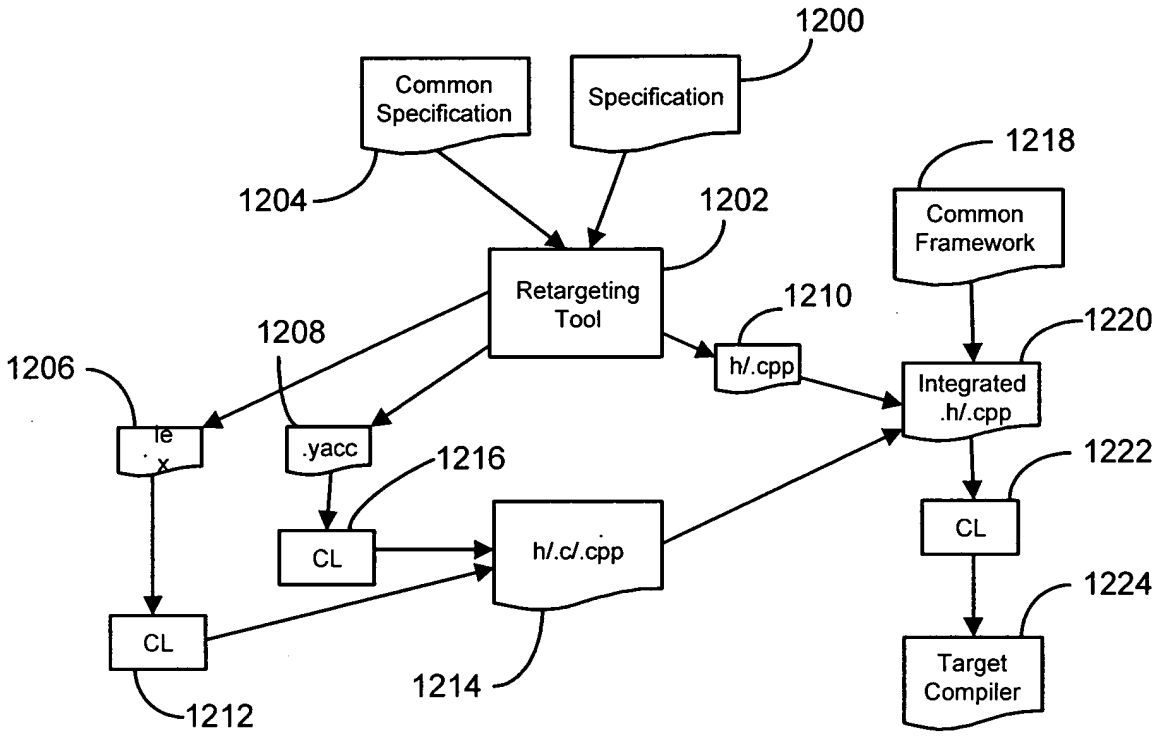

1200
Common Specification
Specification
1204
1202
1218
Common Framework
Retargeting Tool
1210
1208
1220
1206
h/.cpp
Integrated .h/.cpp
le
.x
.yacc
1216
1222
CL
h/.c/.cpp
CL
1224
CL
Target Compiler
1214
1212

FIG. 13

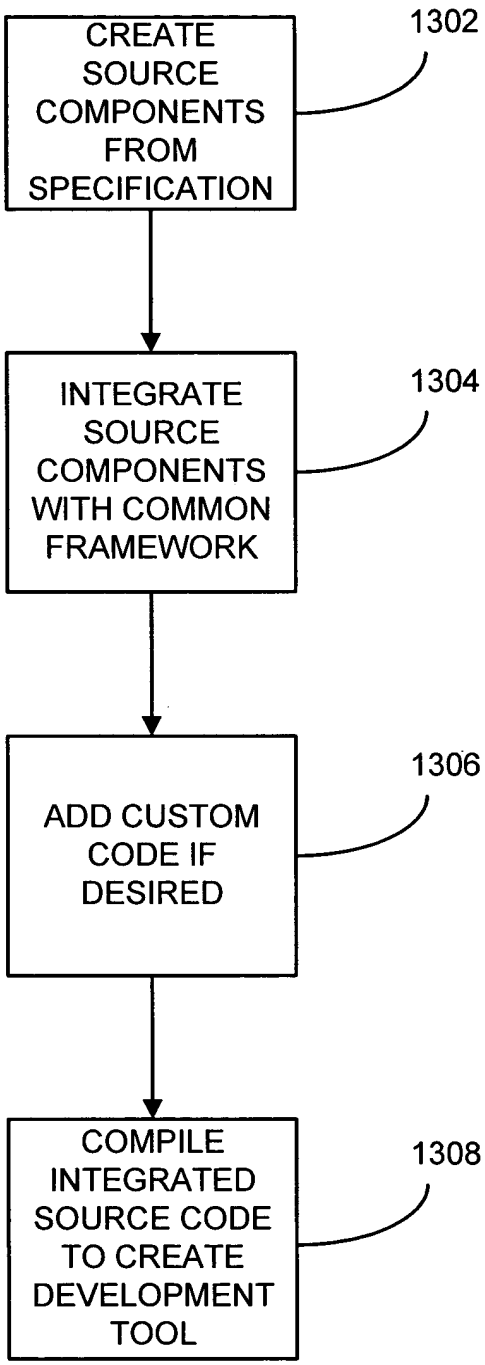

CREATE SOURCE COMPONENTS FROM SPECIFICATION
1302
INTEGRATE SOURCE COMPONENTS WITH COMMON FRAMEWORK
1304
ADD CUSTOM CODE IF DESIRED
1306
COMPILE INTEGRATED SOURCE CODE TO CREATE DEVELOPMENT TOOL
1308

FIG. 14(a)

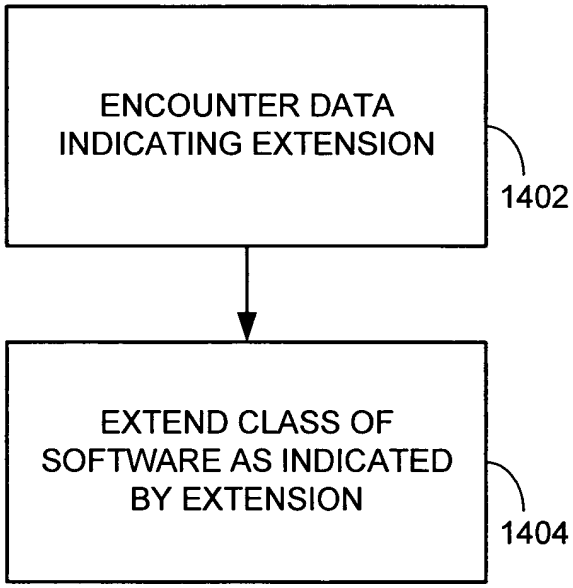

ENCOUNTER DATA INDICATING EXTENSION
1402
EXTEND CLASS OF SOFTWARE AS INDICATED BY EXTENSION
1404

FIG. 14(b)

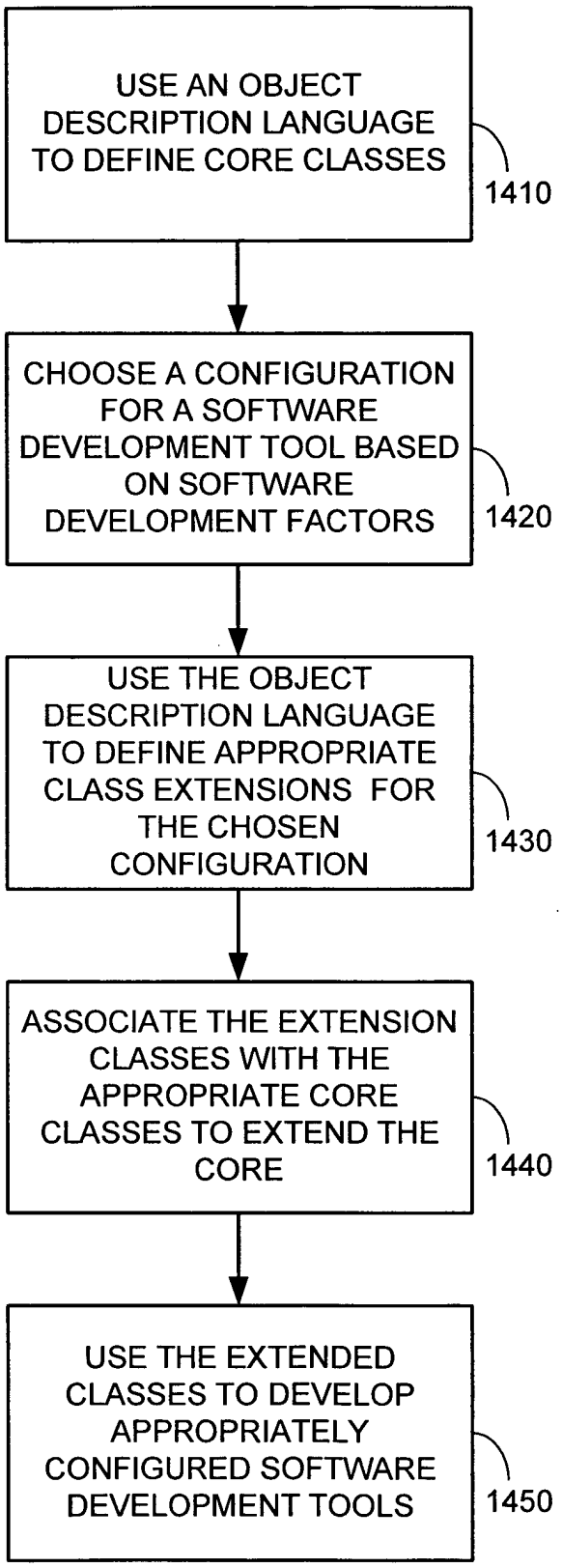

USE AN OBJECT DESCRIPTION LANGUAGE TO DEFINE CORE CLASSES
1410
CHOOSE A CONFIGURATION FOR A SOFTWARE DEVELOPMENT TOOL BASED ON SOFTWARE DEVELOPMENT FACTORS
1420
USE THE OBJECT DESCRIPTION LANGUAGE TO DEFINE APPROPRIATE CLASS EXTENSIONS FOR THE CHOSEN CONFIGURATION
1430
ASSOCIATE THE EXTENSION CLASSES WITH THE APPROPRIATE CORE CLASSES TO EXTEND THE CORE
1440
USE THE EXTENDED CLASSES TO DEVELOP APPROPRIATELY CONFIGURED SOFTWARE DEVELOPMENT TOOLS
1450

FIG. 15(a)

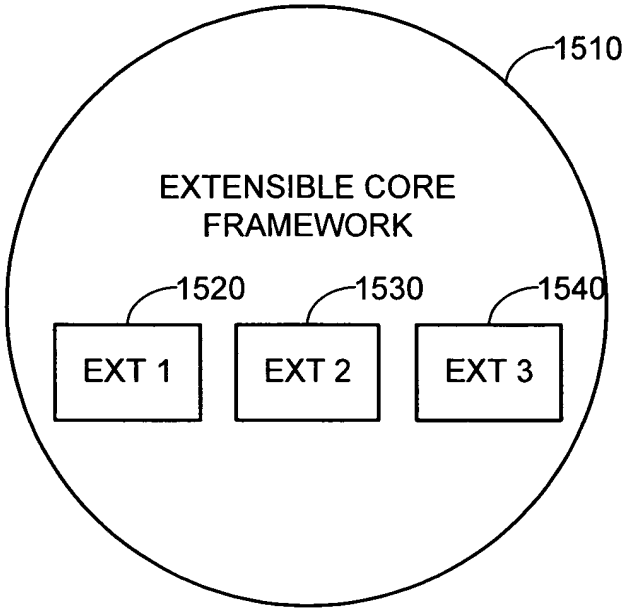


FIG. 15(b)

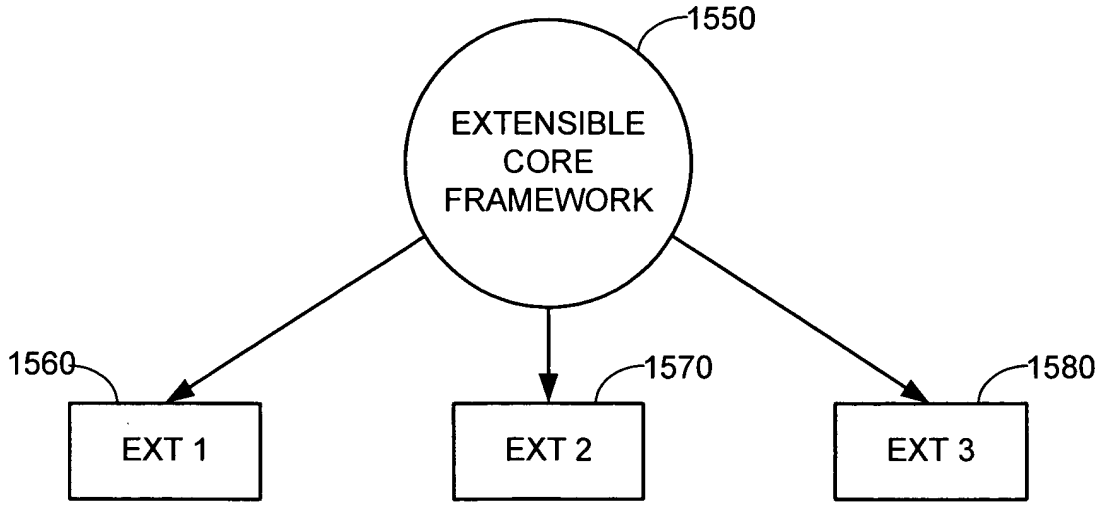
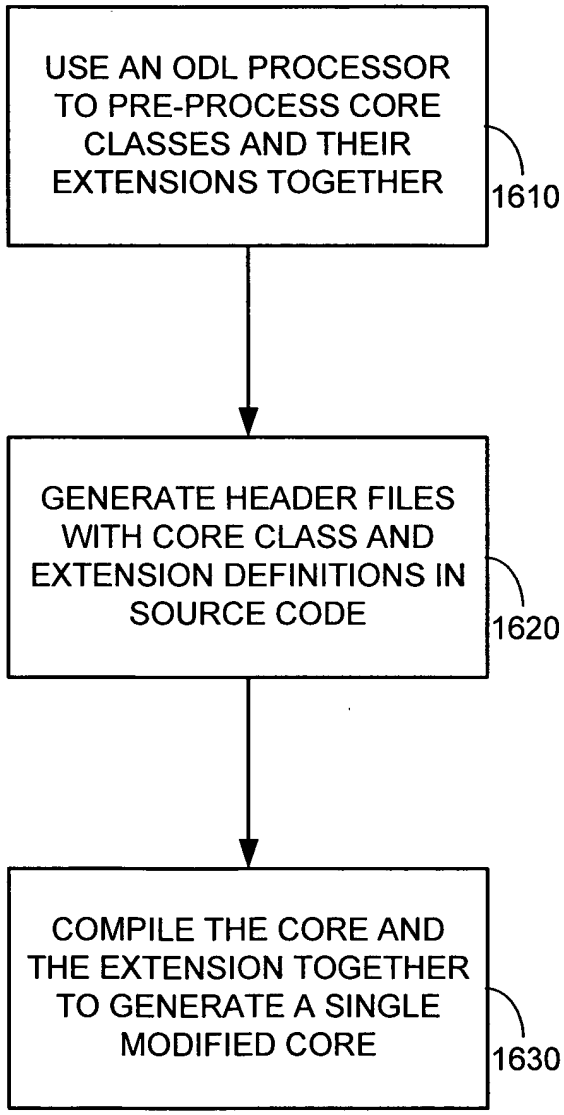


FIG. 16


USE AN ODL PROCESSOR TO PRE-PROCESS CORE CLASSES AND THEIR EXTENSIONS TOGETHER
1610
GENERATE HEADER FILES WITH CORE CLASS AND EXTENSION DEFINITIONS IN SOURCE CODE
1620
COMPILE THE CORE AND THE EXTENSION TOGETHER TO GENERATE A SINGLE MODIFIED CORE
1630

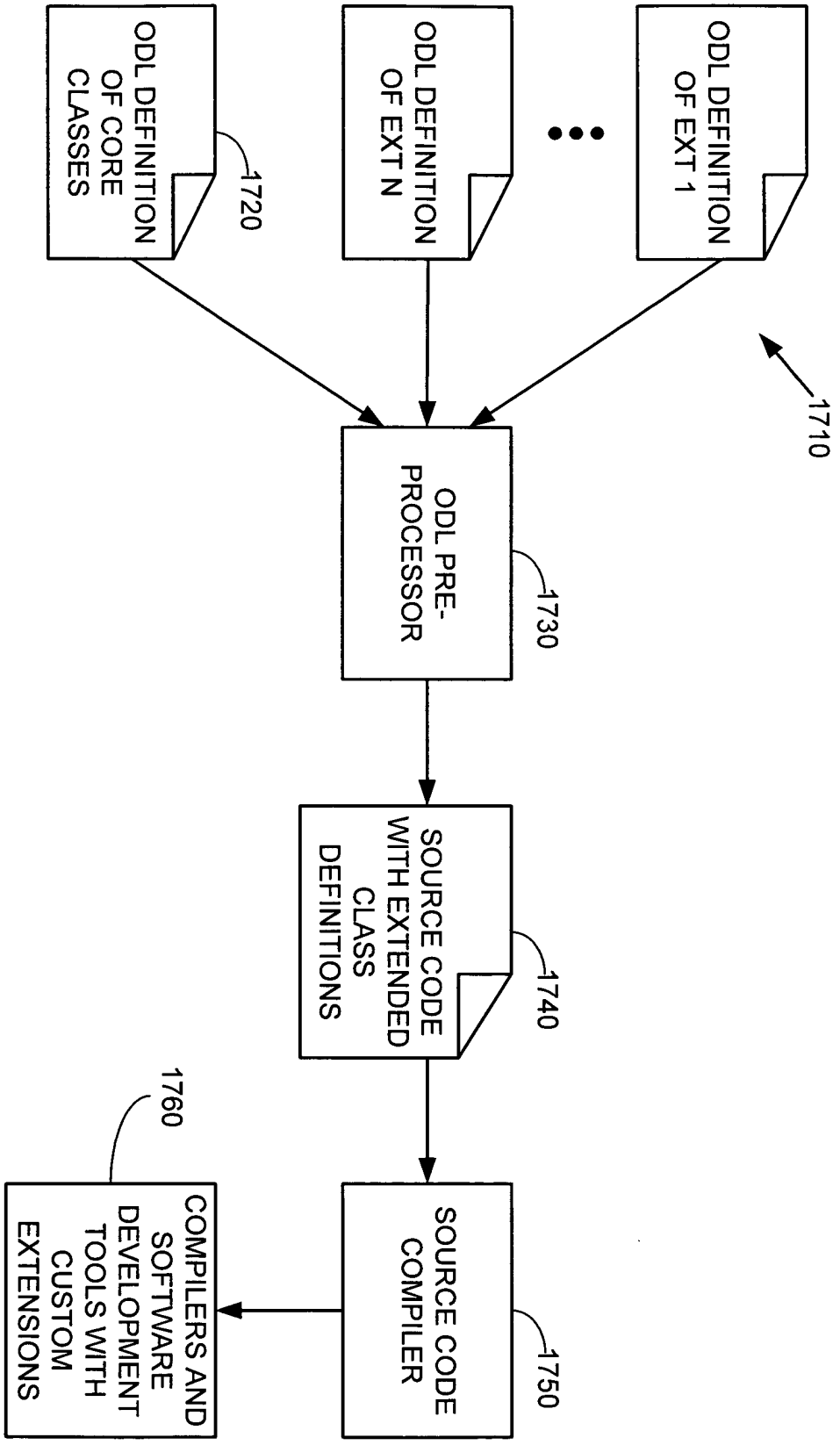
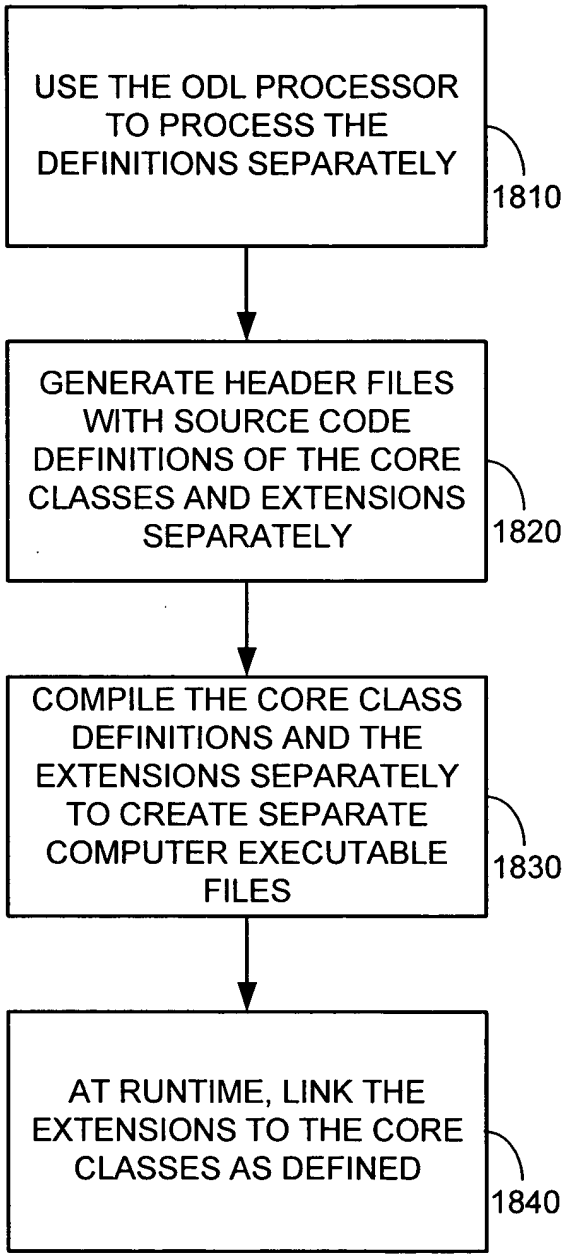

ODL DEFINITION OF CORE CLASSES
1720
ODL DEFINITION OF EXT N
ODL DEFINITION OF EXT 1
1710
ODL PRE-PROCESSOR
1730
SOURCE CODE WITH EXTENDED CLASS DEFINITIONS
1740
1760
COMPILERS AND SOFTWARE DEVELOPMENT TOOLS WITH CUSTOM EXTENSIONS
SOURCE CODE COMPILER
1750

FIG. 17

FIG. 18


USE THE ODL PROCESSOR TO PROCESS THE DEFINITIONS SEPARATELY
1810
GENERATE HEADER FILES WITH SOURCE CODE DEFINITIONS OF THE CORE CLASSES AND EXTENSIONS SEPARATELY
1820
COMPILE THE CORE CLASS DEFINITIONS AND THE EXTENSIONS SEPARATELY TO CREATE SEPARATE COMPUTER EXECUTABLE FILES
1830
AT RUNTIME, LINK THE EXTENSIONS TO THE CORE CLASSES AS DEFINED
1840

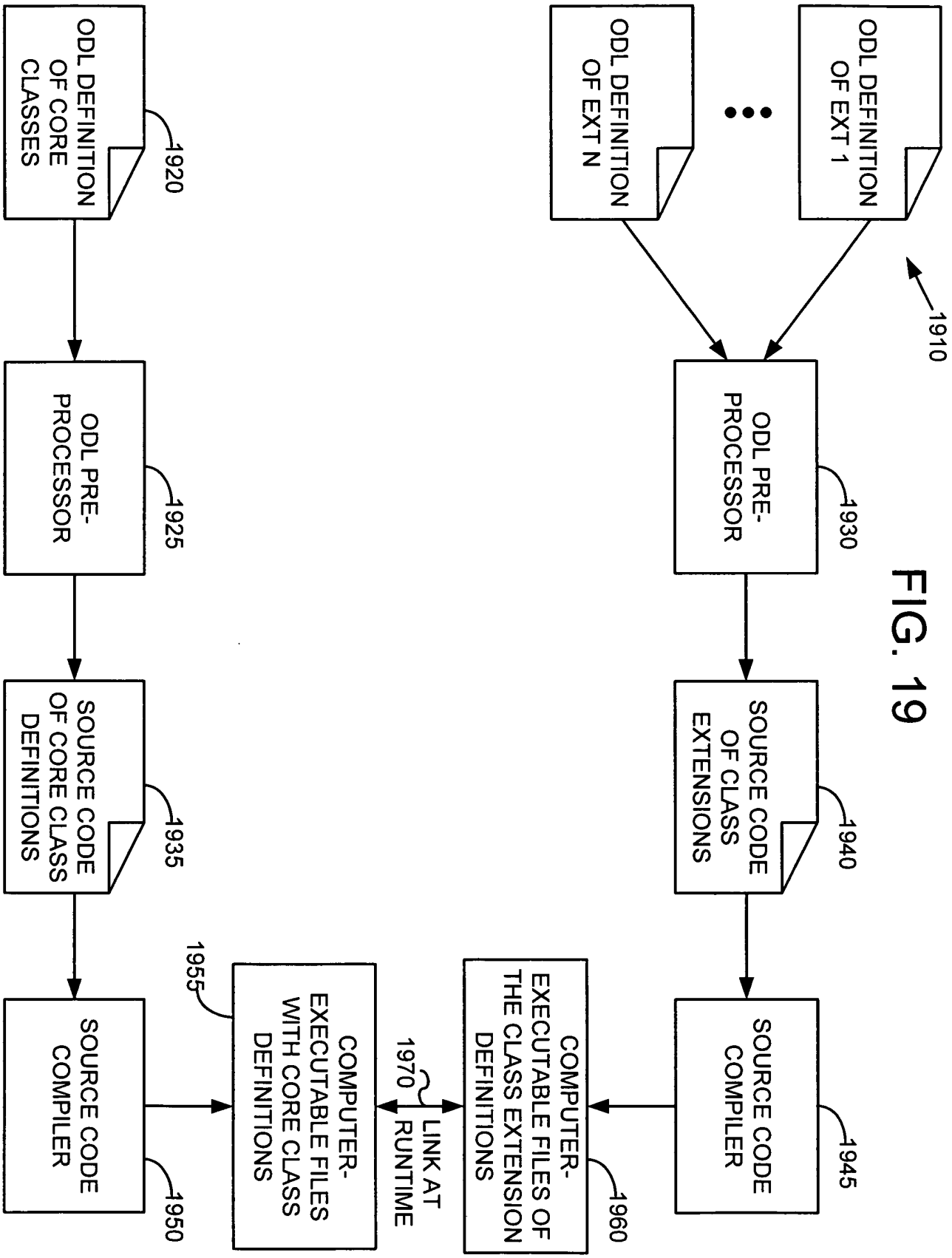

ODL DEFINITION OF EXT 1
ODL DEFINITION OF EXT N
1910
ODL DEFINITION OF CORE CLASSES
1920
ODL PRE-PROCESSOR
1930
ODL PRE-PROCESSOR
1925
SOURCE CODE OF CLASS EXTENSIONS
1940
SOURCE CODE OF CORE CLASS DEFINITIONS
1935
SOURCE CODE COMPILER
1945
SOURCE CODE COMPILER
1950
COMPUTER-EXECUTABLE FILES WITH CORE CLASS DEFINITIONS
1955
LINK AT RUNTIME
1970
COMPUTER-EXECUTABLE FILES OF THE CLASS EXTENSION DEFINITIONS
1960

FIG. 19

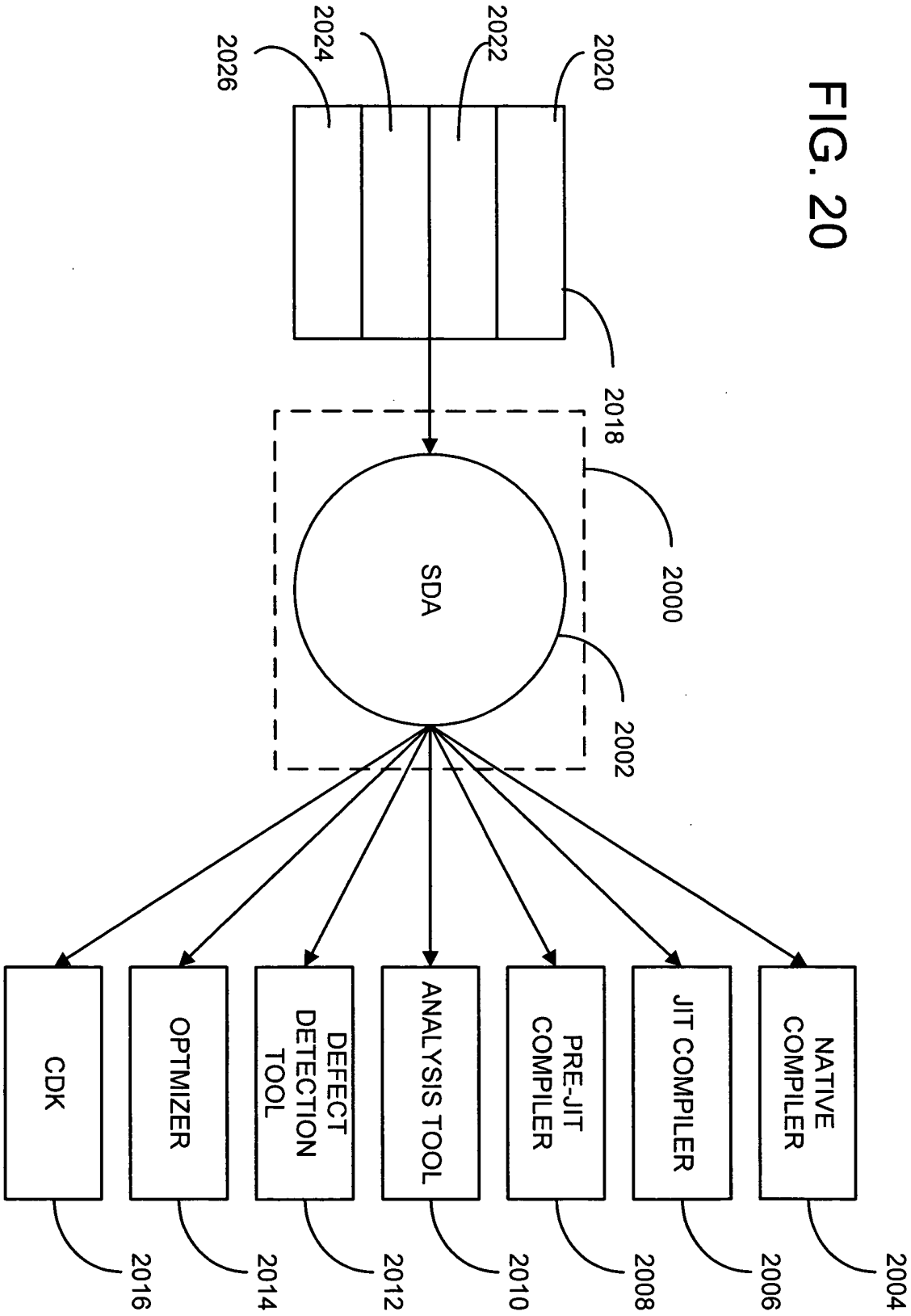

2026
2024
2022
2020
2018
2000
SDA
2002
CDK
OPTMIZER
DEFECT DETECTION TOOL
ANALYSIS TOOL
PRE-JIT COMPILER
JIT COMPILER
NATIVE COMPILER
2016
2014
2012
2010
2008
2006
2004

FIG. 20

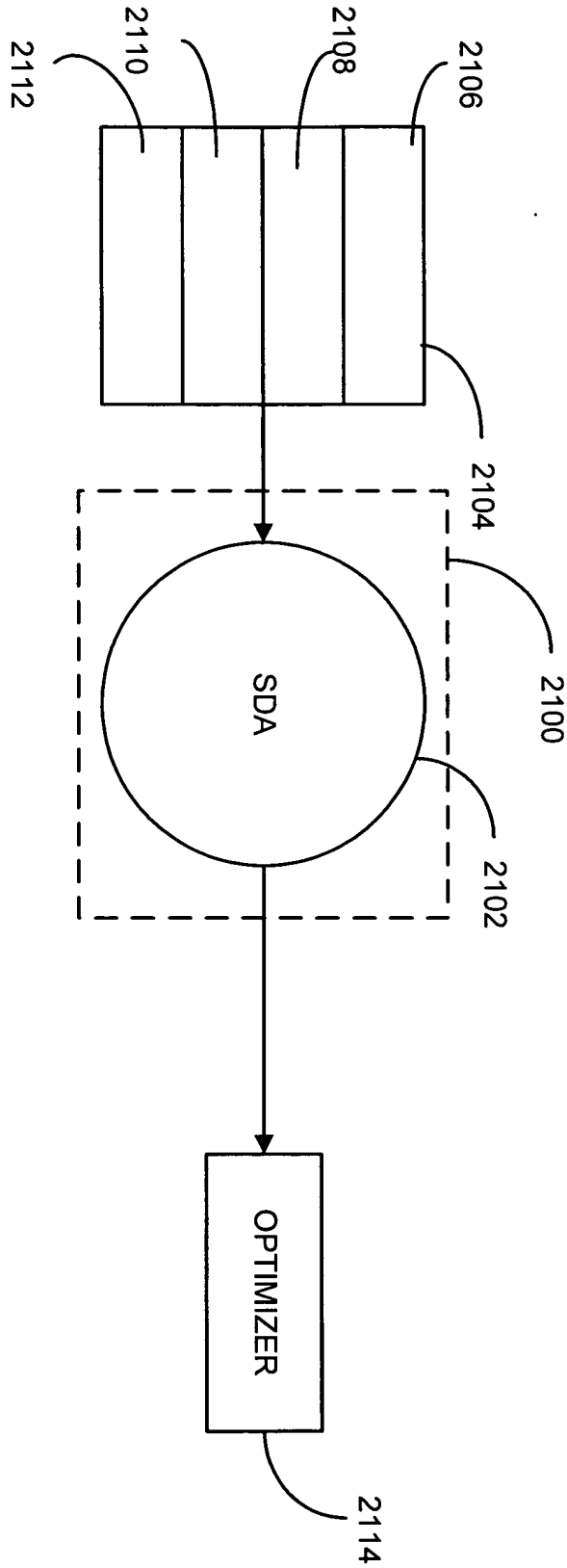


FIG. 21

FIG. 22

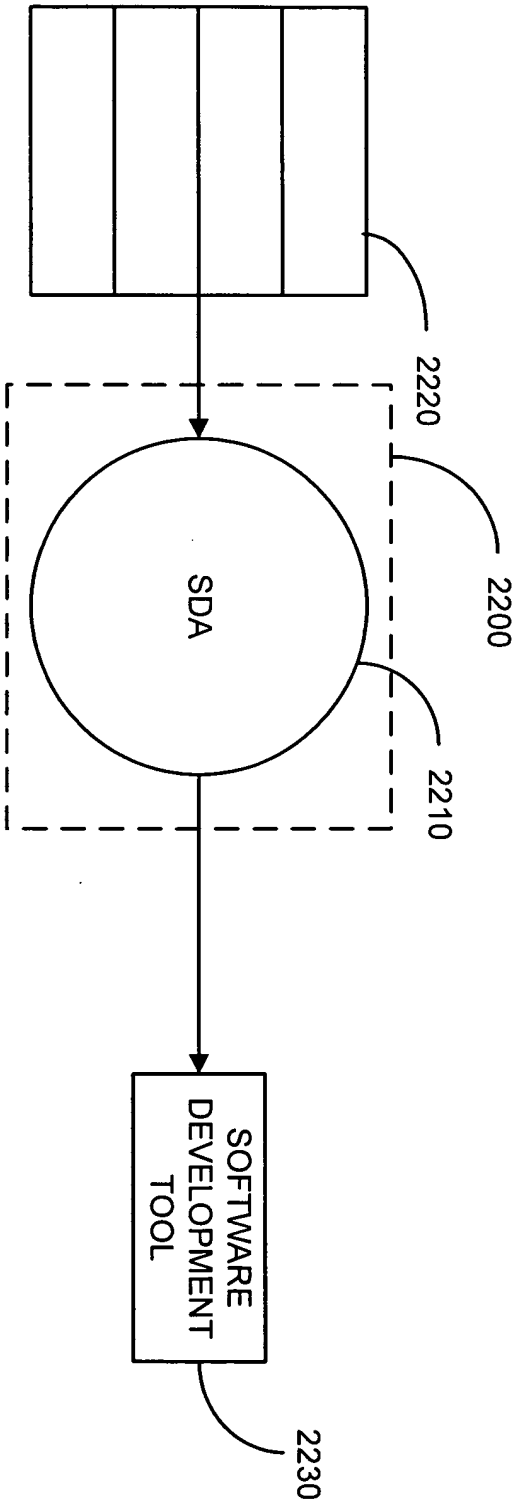


FIG. 23

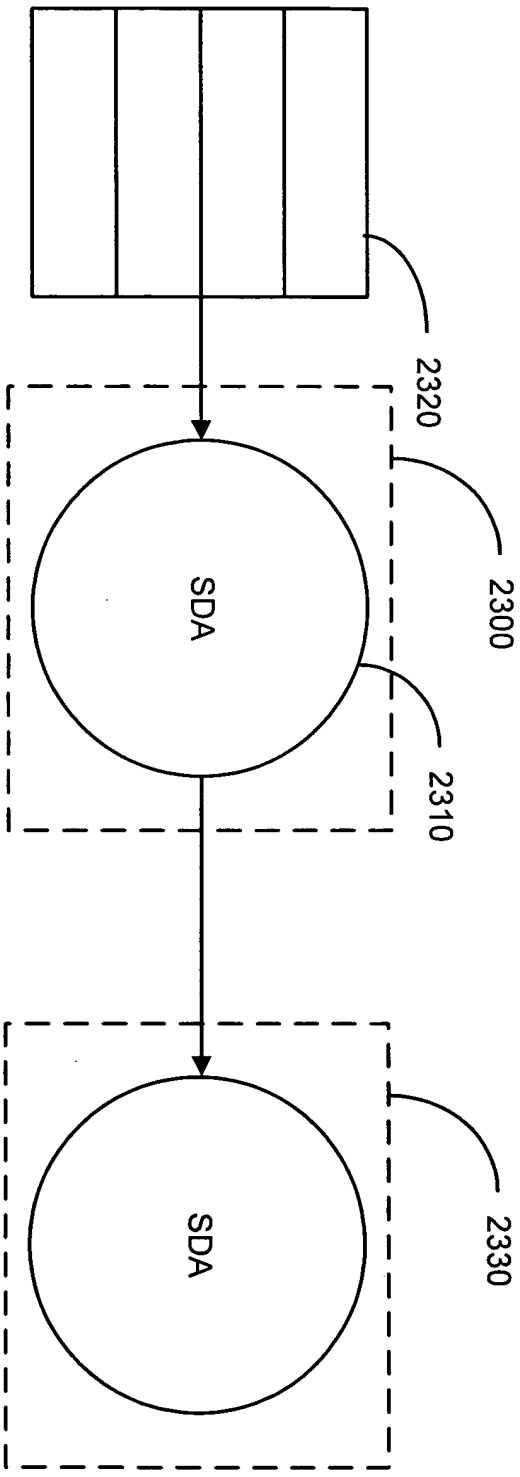


FIG. 24

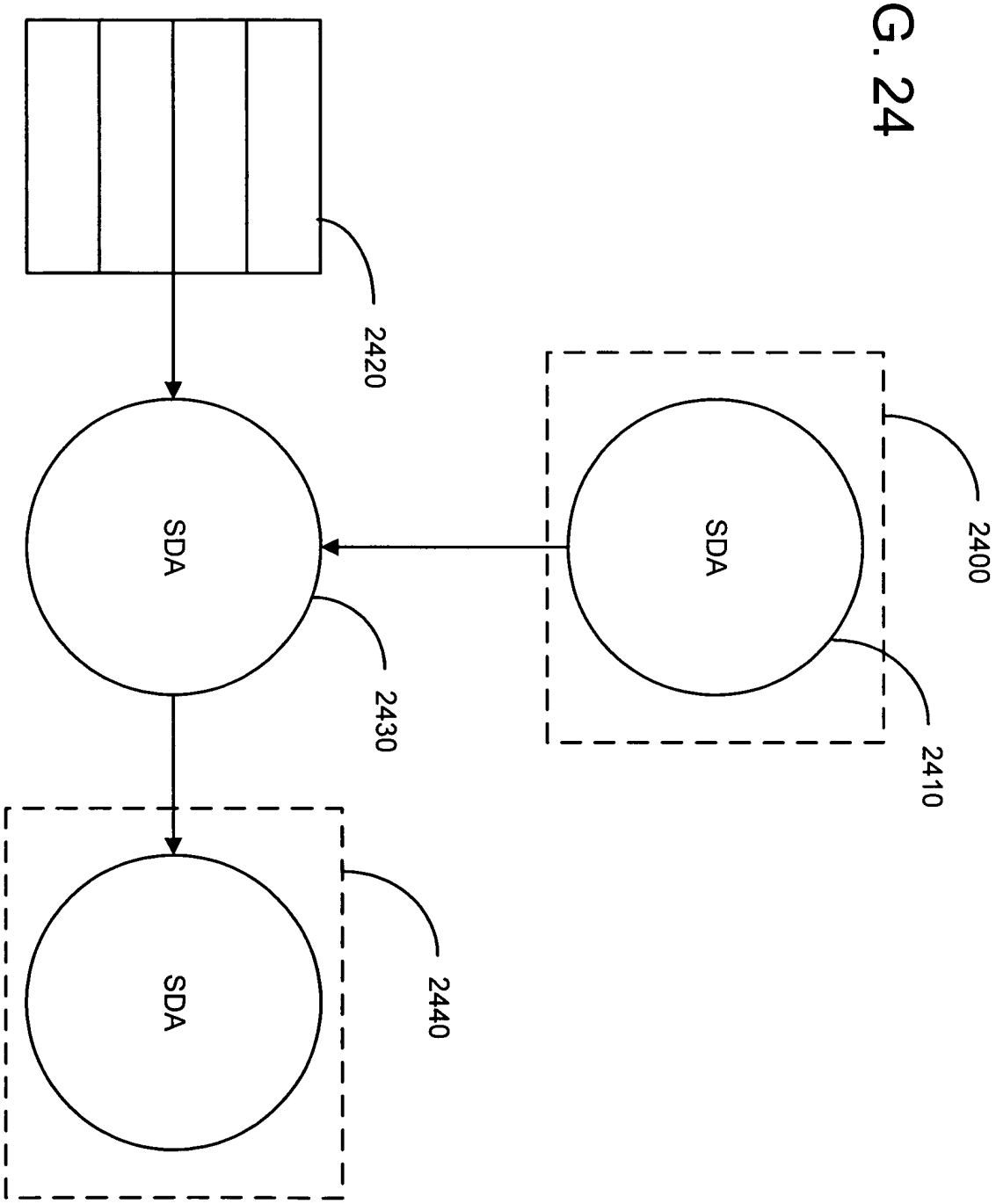
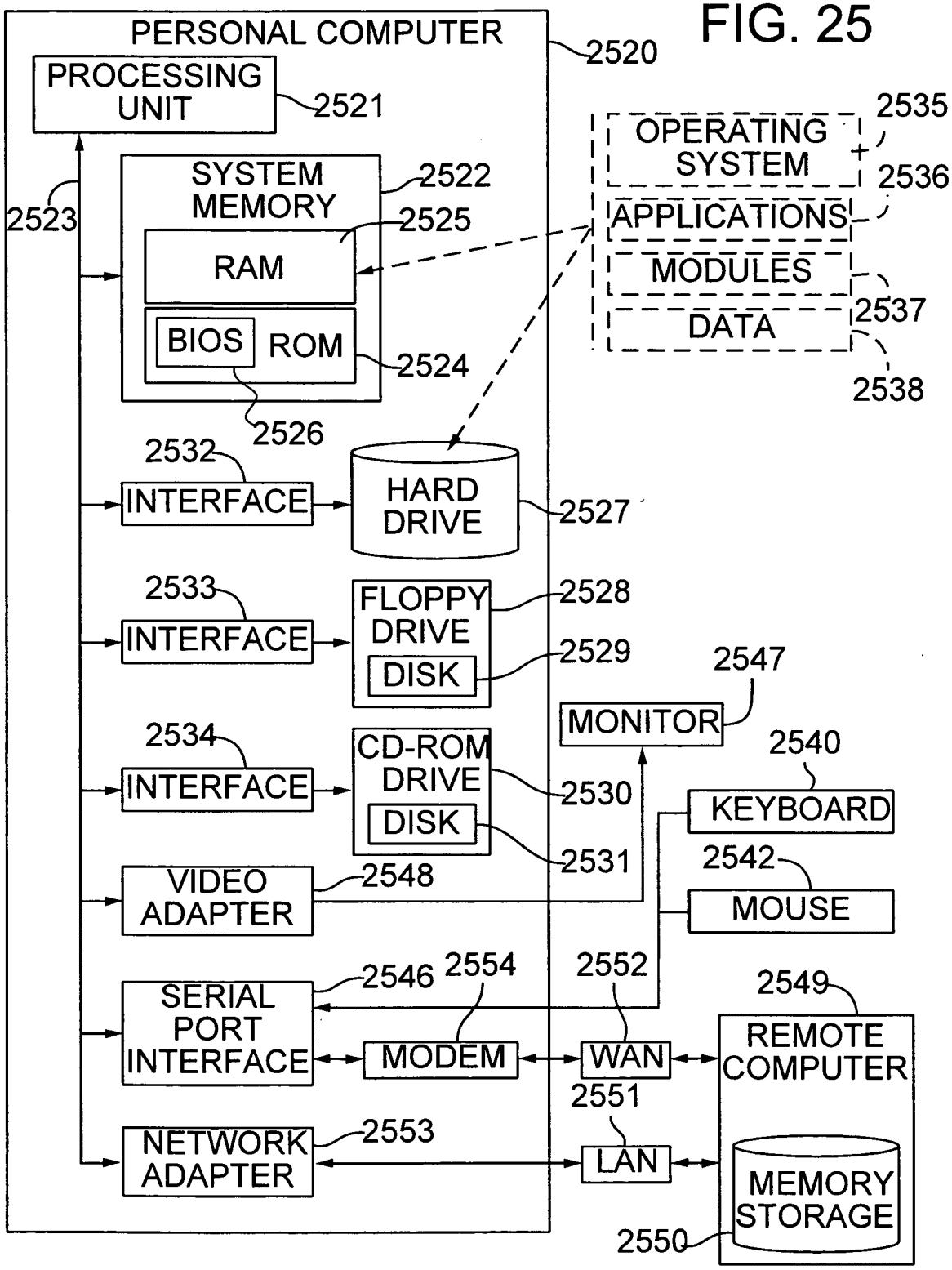


FIG. 25


PERSONAL COMPUTER
2520
PROCESSING UNIT
2521
SYSTEM MEMORY
2522
2523
RAM
2525
BIOS
ROM
2524
2526
OPERATING SYSTEM
2535
2536
APPLICATIONS
MODULES
DATA
2537
2538
2532
INTERFACE
HARD DRIVE
2527
2533
INTERFACE
FLOPPY DRIVE
2528
DISK
2529
2534
INTERFACE
CD-ROM DRIVE
2530
DISK
2531
2547
MONITOR
2540
KEYBOARD
2542
MOUSE
VIDEO ADAPTER
2548
SERIAL PORT INTERFACE
2546
2554
2552
MODEM
WAN
2549
REMOTE COMPUTER
2551
NETWORK ADAPTER
2553
LAN
MEMORY STORAGE
2550